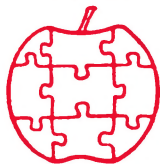


Apple

\$1.80



Assembly

Line

Volume 4 -- Issue 11

August, 1984

In This Issue...

18-Digit Arithmetic, Part 4.	2
Enable/Disable IRQ from Applesoft.	13
Line Number Cross Reference.	15
Speaking of Slow Chips	27
Modify DOS 3.3 for Big BSAVES.	28

New Cross Assemblers Available

We have recently added three new cross assemblers to the S-C Macro family.

Intel Z-8.....	\$32.50
General Instruments 1650....	\$50.00
General Instruments 1670....	\$50.00

Unlike previous cross assemblers, which were based on Version 1.0 of the S-C Macro Assembler, these are based on Version 1.1. This means 80-column support for the Vindex, STB-80, and Apple //e-//c 80-column, as well as standard 40-column. It also adds certain directives and fixes some problems which were in version 1.0.

We have also been hard at work generating Version 2.0 of the S-C Macro Assembler. It will be ready soon, complete with a brand new manual. It will support all the new opcodes and address modes of the 65C02, 65802, and 65816 processors. Owners of older versions of the S-C Assemblers will be able to upgrade for a very reasonable fee.

This month we will look at two output conversion routines. The first one always prints in exponential form, while the second one allows setting a field width and number of fractional digits. The routines are written so that the output string may either be printed or fed to an Applesoft string variable.

Let's assume that the value to be printed has already been loaded into the DP18 accumulator, DAC. Lines 1230-1270 describe DAC as a 12-byte variable. The exponent is in the first byte, DAC.EXPONENT. It has a value from \$00 to \$7F:

\$00 means the whole number is zero
\$01 means the power of ten exponent is -63
\$3F means 10^{-1}
\$40 means 10^0
\$41 means 10^1
\$7F means 10^{63}

The 18 digits of the number, plus two extension digits, are in the next ten bytes in decimal format (each digit takes four bits). The extension is zeroed when you load a fresh value into DAC, but after some computations it holds two more digits to guard against roundoff and truncation problems.

The sign of the number is stored in DAC.SIGN: if the value in DAC.SIGN is from \$00 to \$7F, the number is positive; if from \$80 to \$FF, the number is negative.

If you have been following the DP18 series from the beginning, and typing in all the code (or getting it from the Quarterly Disks), then you will realize that to integrate each installment takes some work. In order to print the sections separately, and have them separately readable, I must repeat some variable declarations. The listing this month refers to two previously printed subroutines, DADD and MOVE.YA.ARG. These are simply equated to \$FFFF in lines 1030 and 1040, so that the code will assemble. If you really want it to work, you have to remove those two lines and include the code for the subroutines. The fact that three installments have already been printed also somewhat restricts me, because even if I see possible improvements I must be careful not to contradict the code you already have.

Quick Standard Format Conversion

The subroutine QUICK.FOUT which begins on line 1600 converts the contents of DAC to a string in FOUT.BUF in the format

sd.dddddddddddddddEsxx

The first s is the sign, which is included only if negative. The d's are a series of up to 18 significant digits (trailing zeroes will not be included). The letter E is always included, to signify the power-of-ten exponent field. The letter s after the E is the sign of the exponent: it is always included, and

S-C Macro Assembler Version 1.0.....\$80
 S-C Macro Assembler Version 1.1.....\$92.50
 Version 1.1 Update.....\$12.50
 Source Code for Version 1.1 (on two disk sides).....\$100
 Full Screen Editor for S-C Macro (with complete source code).....\$49
 S-C Cross Reference Utility (without source code).....\$20
 S-C Cross Reference Utility (with complete source code).....\$50
 DISASM Dis-Assembler (RAK-Ware).....\$30
 Source Code for DISASM.....additional \$30

S-C Word Processor (with complete source code).....\$50
 Double Precision Floating Point for Applesoft (with source code).....\$50
 S-C Documentor (complete commented source code of Applesoft ROMs).....\$50
 Source Code of //e CX & P8 ROMs on disk.....\$15

(All source code is formatted for S-C Macro Assembler Version 1.1. Other assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.....each \$15
 Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.
 QD#1: Oct-Dec 1980 QD#2: Jan-Mar 1981 QD#3: Apr-Jun 1981
 QD#4: Jul-Sep 1981 QD#5: Oct-Dec 1981 QD#6: Jan-Mar 1982
 QD#7: Apr-Jun 1982 QD#8: Jul-Sep 1982 QD#9: Oct-Dec 1982
 QD#10: Jan-Mar 1983 QD#11: Apr-Jun 1983 QD#12: Jul-Sep 1983
 QD#13: Oct-Dec 1983 QD#14: Jan-Mar 1984 QD#15: Apr-Jun 1984

AWIIe Toolkit (Don Lancaster, Synergetics).....\$39
 Quick-Trace (Anthro-Digital).....(reg. \$50) \$45
 Visible Computer: 6502 (Software Masters).....(reg. \$50) \$45
 ES-CAPE: Extended S-C Applesoft Program Editor.....\$60
 Amper-Magic (Anthro-Digital).....(reg. \$75) \$65
 Amper-Magic Volume 2 (Anthro-Digital).....(reg. \$35) \$30
 "Bag of Tricks", Worth & Lechner, with diskette.....(\$39.95) \$36
 Aztec C Compiler System (Manx Software).....(reg. \$199) \$180

Blank Diskettes (Verbatim).....2.50 each, or package of 20 for \$45
 (Premium quality, single-sided, double density, with hub rings)
 Vinyl disk pages, 6"x8.5", hold two disks each.....10 for \$6
 Diskette Mailing Protectors (hold 1 or 2 disks).....40 cents each
 or \$25 per 100

These are cardboard folders designed to fit into 6"x9" Envelopes.
 Envelopes for Diskette Mailers.....6 cents each
 ZIF Game Socket Extender (Ohm Electronics)\$20
 QuikLoader EPROM System (SCRG).....(\$179) \$170

Books, Books, Books.....compare our discount prices!
 "Apple II Circuit Description", Gayler.....(\$22.95) \$21
 "Understanding the Apple II", Sather.....(\$22.95) \$21
 "Enhancing Your Apple II, vol. 1", Lancaster.....(\$15.95) \$15
 Second edition, with //e information.
 "Assembly Cookbook for the Apple II/IIe", Lancaster.....(\$21.95) \$20
 "Incredible Secret Money Machine", Lancaster.....(\$7.95) \$7
 "Beneath Apple DOS", Worth & Lechner.....(\$19.95) \$18
 "What's Where in the Apple", Second Edition.....(\$19.95) \$19
 "6502 Assembly Language Programming", Leventhal.....(\$18.95) \$18
 "6502 Subroutines", Leventhal.....(\$18.95) \$18
 "Real Time Programming - Neglected Topics", Foster.....(\$9.95) \$9
 "Microcomputer Graphics", Myers.....(\$12.95) \$12
 "Microcomputer Design & Troubleshooting", Zumchak.....(\$17.95) \$17

We have small quantities of other great books, call for titles & prices.
 Add \$1.50 per book for US shipping. Foreign orders add postage needed.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
 *** (214) 324-2050 ***
 *** We accept Master Card, VISA and American Express ***

will be either + or -. The xx is a two-digit exponent, and both digits will always be included. The decimal point will be included only if there are non-zero digits after it. If the number is exactly zero, the string in FOUT.BUF will be simply "0". Here are some more examples:

value	string
1000	"1E+03"
.001	"1E-03"
-262564478.5	"-2.625644785E+08"

Two processes are involved in converting from DAC to FOUT.BUF. One is the analysis of the DAC contents; the other is the process of storing sequential characters into FOUT.BUF. The latter process is handled in most cases by the subroutine at lines 3720-3820. Entry at STORE.CHAR stores the contents of the A-register in the next position in FOUT.BUF, and increments the position pointer (INDEX). Entry at STORE.DIGIT first converts the value in the A-register to an ASCII digit by setting the high nybble to "3". (The digits 0-9 are \$30-\$39 in ASCII.)

QUICK.FOUT begins by setting INDEX, the FOUT.BUF position pointer, to 0. At lines 1630-1700 the special case of the value in DAC being exactly zero is tested and handled. If the value in DAC is zero, then DAC.EXPONENT will be zero. (This is a convention throughout DP18, to simplify making values of zero and testing for them.) If the value is zero, ASCII zero is stored in FOUT.BUF, followed by a terminating \$00 byte.

If the value is not zero, the next job is to check the sign of the value. Lines 1710-1740 insert a minus sign in FOUT.BUF if the value is negative.

Lines 1760-1910 pull out the 18 digits of the mantissa from DAC.HI through DAC.HI+8. The extension digits are ignored. The code here looks an awfully lot like a routine to convert from hex to ASCII, ignoring the possible hex digits A-F. That is because the digits are four bits each, and ARE like hex digits. Lines 1830-1860 insert the decimal point after the first digit.

Lines 1930-2020 look at the formatted number in FOUT.BUF and trim off the trailing zeroes. If all digits after the decimal point are zero, the decimal point is trimmed off too. If you would rather that QUICK.FOUT always printed exactly 18 digits, trailing zeroes and all, you could cut out these lines.

Lines 2040-2290 format the exponent field. First the letter E is installed in FOUT.BUF. Then lines 2060-2120 install the exponent sign. There is a little adjustment here due to the fact that the value in DAC is in the form ".DDDD" times a power of ten, and we are converting to "D.DDD" times a power. That means the exponent in DAC.EXPONENT is one larger than we will print. The DEY at line 2080 adjusts for this offset.

Lines 2130-2180 get the absolute value of the exponent by removing the \$40 bias and taking the 2's complement if the

result is negative. Lines 2190-2290 convert the binary value of the exponent to two decimal digits, and insert them into FOUT.BUF. Lines 2300-2310 terminate the FOUT.BUF string with \$00.

Once the value has been converted to a string in FOUT.BUF, we can either print it or put it into an Applesoft string variable. The subroutine QUICK.PRINT which begins at line 1370 calls QUICK.FOUT and then prints the characters from FOUT.BUF.

Fancier Formatted Conversion

The second conversion routine, which begins at line 2350, allows you to specify the number of digits to display after the decimal point, and the number of characters in the output field. The value will be formatted into the field against the right end, with leading blanks as necessary to fill the field. The value will be rounded to the number of digits that will be converted. If you are familiar with the FORTRAN language, you will recognize this as the "Fw.d" format. W is the width of the field, and D is the number of fractional digits. Here are some examples:

W	D	value	string
12	5	1234.56	" 1234.56000"
12	1	1234.56	" 1234.6"

As before, the output string will be stored in FOUT.BUF in ASCII code, terminated by a \$00 byte. If the value will not fit into the W.D field you specify, the entire field will be filled with "*" characters.

As listed here, I have set FOUT.BUF as a 41-byte variable. This means the maximum W is 40, leaving room for the terminating \$00 byte. If you want longer conversions, simply change line 1060.

FOUT expects the W and D parameters to be in the A- and Y-registers, respectively. Lines 2380-2460 check W and D for legality. If W is larger than FOUT.BUF.SIZE-1, then it is set to that value. We don't want to store converted characters beyond the end of FOUT.BUF! Then if D is larger than W-1, it is pruned back.

Lines 2480-2540 initialize various variables used during the following conversion. Once again, INDEX will point to the position in FOUT.BUF. I could probably have economized some in the use of variables by re-using the same variables for different purposes, but I wanted to keep them separate to make it easier to code and debug.

Line 2560 calls ROUND.DAC.D to round the value in DAC to D digits after the decimal point. This boils down to adding .5 times 10 to the D power to the value in DAC. ROUND.DAC.D, at lines 3860-4000, does just that. First the rounding number is built in ARG, then DADD adds ARG to FAC.

Lines 2570-2610 store a minus sign into SIGN.CHAR if the value in DAC is negative. SIGN.CHAR was initialized to \$00 above. If the sign is negative, line 2590 will increment SIGN.SIZE. SIGN.SIZE will either be 0 or 1, and will be used later in determining how many leading blanks are needed. We cannot store the sign character into FOUT.BUF until the leading blanks have been stored.

Lines 2630 to 2710 compute how many digits will be printed before the decimal point (NO.LEADING.DIGITS), and how many zeroes before the first significant digit after the decimal point (NO.LEADING.ZEROES). If the power-of-ten exponent was negative, there will be no leading digits and some leading zeroes; if positive, there will be some leading digits and no leading zeroes. For example,

.2345E-5	.000002345	5 leading zeroes
.2345E+3	234.5	3 leading digits

What if the exponent is more than 18? This would mean more digits might be extracted from DAC than exist, so lines 2730-2790 limit NO.LEADING.DIGITS to 18. NO.INTEGRAL.ZEROES takes up the slack, to print any necessary zeroes between the last significant digit before the decimal point, and the decimal point. For example, if W=25 and D=2, and the value is .1234E+20, we will get NO.LEADING.DIGITS=18 and NO.INTEGRAL.ZEROES=2:

" 12340000000000000000.00"

Lines 2810-2870 now calculate the total number of non-blank characters which will be required: one for sign if the sign is negative, all the leading digits and integral zeroes before the decimal point, one for the decimal point itself, and D fractional digits. (Just now I noticed that I could have saved two bytes and two cycles by changing line 2810 from CLC to SEC, and eliminating the ADC #1 at line 2860.)

Lines 2890-2920 compute how many significant digits of fraction will be needed. You specified D digits of fraction, but only DD of them will come from the value in DAC. This will be less than D if there are any leading zeroes.

Lines 2940-2970 check whether the converted number can fit in a W-wide field. If not, Lines 3370-3400 fill the field with stars and exit.

Lines 2980-3030 compute how many leading blanks will be needed to right justify the number in the W-field. There is some hopscotch here because we are going to put "0." in front of numbers that have no integral digits.

At long last, we are ready to begin string characters in FOUT.BUF. Lines 3050-3070 store the leading blanks. A subroutine STORE.N.CHARS does the dirty work. STORE.N.CHARS (lines 3670-3710) expects the character to be stored in the A-register, and the count in the Y-register. It also expects that the Z-status is set according to the count in Y. Thus, if



FONT DOWNLOADER & EDITOR (\$39.00)

Turn your printer into a custom typesetter. Downloaded characters remain active while printer is powered. Use with any Word Processor program capable of sending ESC and control codes to printer. Switch back and forth easily between standard and custom fonts. All special printer functions (like expanded, compressed etc.) apply to custom fonts. Full HIRES screen editor lets you create your own characters and special graphics symbols. Compatible with many parallel printer I/F cards. User driver option provided. For Apple II, II+, //e. Specify printer: Apple Dot Matrix, C.Itoh 8510A (Prowriter), Epson FX 80/100, or OkiData 92/93.

NEW !!! The Font Downloader & Editor for the Apple Imagewriter Printer. For use with Apple II, II+, //e (with SuperSerial card) and the new Apple //c (with builtin serial interface).

NEW !!! FONT LIBRARY DISKETTE #1 (\$19.00) contains lots of user-contributed fonts for all printers supported by the Font Downloader & Editor. Specify printer with order.

DISASM 2.2e - AN INTELLIGENT DISASSEMBLER (\$30.00)

Investigate the inner workings of machine language programs. DISASM converts machine code into meaningful, symbolic source. Creates a standard text file compatible with S-C, LISA, ToolKit and other assemblers. Handles data tables, displaced object code & even lets you substitute your own meaningful labels. (100 commonly used Monitor and Pg Zero names included.) An address-based triple cross reference table is provided to screen or printer. DISASM is an invaluable machine language learning aid to both novice & expert alike. Don Lancaster says DISASM is "absolutely essential" in his new ASSEMBLY COOKBOOK. For entire Apple II family including the new Apple //c (with all the new opcodes). **SOURCE CODE available for an additional \$30.00**

S-C Assembler (Ver 4.0 only) SUPPORT UTILITY PACKAGE (\$30.00)

- * SC.XREF - Generates a GLOBAL LABEL Cross Reference Table for complete documentation of source listings.
- * SC.GSR - Global Search & Replace eliminates tedious manual renaming of labels. Search all/part of source.
- * SC.TAB - Tabulates source files into neat, readable form. **SOURCE CODE available for an additional \$30.00**

The 'PERFORMER' CARD (\$39.00)

Plugs into any slot to convert a 'dumb' centronics-type printer I/F card into a 'smart' one. Command menu eliminates need to remember complicated ESC codes. Features include perforation skip, auto page numbering with date & title. Includes large HIRES graphics & text screen dumps. Specify printer: MX-80 with Graftrax-80, MX-100, MX-80/100 with Graftraxplus, NEC 8092A, C.Itoh 8510 (Prowriter), OkiData 82A/83A with Okigraph & OkiData 92/93. **SOURCE CODE: \$30.00**

FIRMWARE FOR APPLE-CAT: The 'MIRROR' ROM (\$25.00)

Communications ROM plugs directly into Novation's Apple-Cat Modem card. Basic modes: Dumb Terminal, Remote Console & Programmable Modem. Features include: selectable pulse or tone dialing, true dialtone detection, audible ring detect, ring-back, printer buffer, 80 col card & shift key mod support. Uses superset of Apple's Comm card and Micromodem II commands. **SOURCE CODE: \$50.00**

RAM/ROM DEVELOPMENT BOARD (\$30.00)

Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip (6116 type RAM for program development or 2716 EPROM to keep your favorite routines on-line). Maps into \$Cn00-CnFF and \$C800-CFFF.

NEW !!! C-PRINT For The APPLE //c (\$99.00)

Connect standard parallel printers to an Apple //c. C-PRINT is a hardware accessory that plugs into the standard Apple //c printer serial port. The other end plugs into any printer having a standard 36 pin centronics-type parallel connector. Just plug in and print! High speed data transfer at 9600 Baud. No need to reconfigure serial port or load software drivers for text printing.

Avoid a \$3.00 postage/handling charge by enclosing full payment with order. (Mastercard & VISA excluded)

RAK-WARE 41 Ralph Road W. Orange N J 07052 (201) 325-1885



the count is zero, the subroutines returns immediately without storing any characters.

STORE.N.DIGITS, at lines 3440-3620, is quite similar to STORE.N.CHARS. Once again, the count must be in the Y-register and the Z-status should reflect the value in Y. Digits are picked out of the value in DAC using an index DIGIT.PICKER, and stored into FOUT.BUF using STORE.DIGIT.

Lines 3090-3110 store the sign if it is negative. Lines 3120-3210 print whatever digits are needed before the decimal point. This will include leading digits (if any) and integral zeroes (if any), or simply one zero (if neither of the other).

Lines 3230-3320 store the fractional part. This includes the decimal point, leading fractional zeroes (if any), and fractional digits (if any).

Finally, lines 3340-3350 store a terminating \$00 at the end of the string in FOUT.BUF.

A subroutine called FORMAT.PRINT at line 1450 calls FOUT and then prints the contents of FOUT.BUF. You could now write a higher level routine, if you wish, which would examine the exponent to determine whether the number would fit in a 20-character field. If not, you could use QUICK.PRINT. If so, use FOUT with W=40 and D=18, and then truncate leading spaces and trailing zeroes. This would give you a complete print routine for any numbers, printing them in simple form when they fit and exponential form when they don't. Indeed, just such a routine already exists in DP18, but will have to wait for a future installment. FOUT can also be used as the base for a complete PRINT USING facility, and that is also already in DP18 waiting for future installments.

Meanwhile, enjoy these two conversions, and experiment with your own.

	1000	*SAVE S.DP18 FOUT	
	1010	*-----	
DB5C-	1020	AS.COUT	.EQ \$DB5C
FFFF-	1030	DADD	.EQ \$FFFF
FFFF-	1040	MOVE.YA.ARG	.EQ \$FFFF
	1050	*-----	
0800-	1060	FOUT.BUF	.BS 41
29-	1070	FOUT.BUF.SIZE	.EQ *-FOUT.BUF
	1080	*-----	
0829-	1090	W	.BS 1
082A-	1100	D	.BS 1
082B-	1110	INDEX	.BS 1
082C-	1120	SIGN.SIZE	.BS 1
082D-	1130	SIGN.CHAR	.BS 1
082E-	1140	ZERO.CHAR	.BS 1
082F-	1150	WW	.BS 1
0830-	1160	DD	.BS 1
0831-	1170	DIGIT.PICKER	.BS 1
0832-	1180	NO.LEADING.ZEROS	.BS 1
0833-	1190	NO.LEADING.DIGITS	.BS 1
0834-	1200	NO.INTEGRAL.ZEROS	.BS 1
0835-	1210	NO.LEADING.BLANKS	.BS 1
	1220	*-----	
0836-	1230	DAC	.BS 12
0836-	1240	DAC.EXPONENT	.EQ DAC
0837-	1250	DAC.HI	.EQ DAC+1
0840-	1260	DAC.EXTENSION	.EQ DAC+10
0841-	1270	DAC.SIGN	.EQ DAC+11
	1280	*-----	


```

0842- 1290 ARG .BS 12
0842- 1300 ARG.EXPONENT .EQ ARG
0843- 1310 ARG.HI .EQ ARG+1
084C- 1320 ARG.EXTENSION .EQ ARG+10
084D- 1330 ARG.SIGN .EQ ARG+11
1340 *-----
1350 * QUICK PRINT
1360 *-----
1370 QUICK.PRINT
084E- 20 6A 08 1380 JSR QUICK.FOUT
0851- 4C 5C 08 1390 JMP FOR.PRINT.1
1400 *-----
1410 * FORMATTED PRINT
1420 * (A)=WIDTH OF FIELD
1430 * (Y)=# OF FRACTIONAL DIGITS
1440 *-----
1450 FORMAT.PRINT
0854- A2 30 1460 LDX #'0 USE ZEROES BEFORE FRACTION
0856- 8E 2E 08 1470 STX ZERO.CHAR
0859- 20 F1 08 1480 JSR FOUT
1490 *-----
1500 FOR.PRINT.1
085C- A0 00 1510 LDY #0
085E- B9 00 08 1520 .1 LDA FOUT.BUF,Y
0861- F0 06 1530 BEQ .2
0863- 20 5C DB 1540 JSR AS.COUT
0866- C8 1550 INY
0867- D0 F5 1560 BNE .1 ...ALWAYS
0869- 60 1570 .2 RTS
1580 *-----
1590 * QUICK CONVERSION
1600 *-----
1610 QUICK.FOUT
086A- A0 00 1620 LDY #0
086C- 8C 2B 08 1630 STY INDEX
086F- AD 36 08 1640 LDA DAC.EXPONENT
0872- D0 0C 1650 BNE .0 NUMBER IS NOT ZERO
0874- EE 2B 08 1660 INC INDEX
0877- 8C 01 08 1670 STY FOUT.BUF+1
087A- A9 30 1680 LDA #'0
087C- 8D 00 08 1690 STA FOUT.BUF MAKE IT '0'
087F- 60 1700 RTS
0880- AD 41 08 1710 .0 LDA DAC.SIGN
0883- 10 05 1720 BPL .1
0885- A9 2D 1730 LDA #'- NEGATIVE
0887- 20 01 0A 1740 JSR STORE.CHAR
1750 *-----
088A- B9 37 08 1760 .1 LDA DAC.HI,Y NEXT BYTE OF #
088D- 48 1770 PHA
088E- 4A 1780 LSR
088F- 4A 1790 LSR
0890- 4A 1800 LSR
0891- 4A 1810 LSR
0892- 20 FD 09 1820 JSR STORE.DIGIT
0895- C0 00 1830 CPY #0
0897- D0 05 1840 BNE .2
0899- A9 2E 1850 LDA #1. PUT DECIMAL POINT
089B- 20 01 0A 1860 JSR STORE.CHAR
089E- 68 1870 .2 PLA DO 2ND DIGIT
089F- 20 FD 09 1880 JSR STORE.DIGIT
08A2- C8 1890 INY
08A3- C0 09 1900 CPY #9 8 MORE BYTES
08A5- 90 E3 1910 BCC .1
1920 *-----
08A7- AC 2B 08 1930 LDY INDEX TRUNCATE TRAILING ZEROS
08AA- 88 1940 .3 DEY
08AB- B9 00 08 1950 LDA FOUT.BUF,Y
08AE- C9 30 1960 CMP #'0
08B0- F0 F8 1970 BEQ .3 DONE
08B2- C9 2E 1980 CMP #1. TRAILING DECIMAL PT?
08B4- D0 01 1990 BNE .4 NO
08B6- 88 2000 DEY YES, DELETE IT
08B7- C8 2010 .4 INY
08B8- 8C 2B 08 2020 STY INDEX SAVE NEW END OF NUMBER
2030 *-----
08BB- A9 45 2040 LDA #'E
08BD- 20 01 0A 2050 JSR STORE.CHAR E FOR EXPONENT
08C0- A9 2B 2060 LDA #'+
08C2- AC 36 08 2070 LDY DAC.EXPONENT

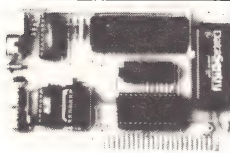
```

08C5-	88		2080	DEY	
08C6-	C0	40	2090	CPY	#\$40
08C8-	B0	02	2100	BCS	.5
08CA-	A9	2D	2110	LDA	#'-
08CC-	20	01	2120	JSR	STORE.CHAR
08CF-	98		2130	TYA	EXPONENT
08D0-	38		2140	SEC	
08D1-	E9	40	2150	SBC	#\$40 REMOVE OFFSET
08D3-	10	04	2160	BPL	.6
08D5-	49	FF	2170	EOR	#\$FF
08D7-	69	01	2180	ADC	#1
08D9-	A2	2F	2190	LDX	#'0-1
08DB-	38		2200	SEC	
08DC-	E8		2210	INX	
08DD-	E9	0A	2220	SBC	#10
08DF-	B0	FB	2230	BCS	.8
08E1-	69	3A	2240	ADC	#'9+1
08E3-	48		2250	PHA	
08E4-	8A		2260	TXA	
08E5-	20	01	2270	JSR	STORE.CHAR
08E8-	68		2280	PLA	
08E9-	20	01	2290	JSR	STORE.CHAR
08EC-	A9	00	2300	LDA	#0
08EE-	4C	01	2310	JMP	STORE.CHAR
			2320		
			2330		*****
			2340		FORMATTED CONVERSION
			2350		(A)=WIDTH OF FIELD
			2360		(Y)=# OF FRACTIONAL DIGITS
			2370		*****
			2380		FOUT
08F1-	C9	29	2380	CMP	#FOUT.BUF.SIZE LIMIT WIDTH
08F3-	90	02	2390	BCC	.1
08F5-	A9	28	2400	LDA	#FOUT.BUF.SIZE-1
08F7-	8D	29	2410	STA	W
08FA-	CC	29	2420	CPY	W FORCE D<W
08FD-	90	02	2430	BCC	.2
08FF-	A8		2440	TAY	
0900-	88		2450	DEY	
0901-	8C	2A	2460	STY	D
			2470		*****
0904-	A9	00	2480	LDA	#0
0906-	8D	2B	2490	STA	INDEX
0909-	8D	2C	2500	STA	SIGN.SIZE
090C-	8D	2D	2510	STA	SIGN.CHAR
090F-	8D	34	2520	STA	NO.INTEGRAL.ZEROES
0912-	8D	32	2530	STA	NO.LEADING.ZEROES
0915-	8D	31	2540	STA	DIGIT.PICKER
			2550		*****
0918-	20	0B	2560	JSR	ROUND.DAC.D ROUND TO D DIGITS
091B-	AD	41	2570	LDA	DAC.SIGN
091E-	10	08	2580	BPL	.3
0920-	EE	2C	2590	INC	SIGN.SIZE
0923-	A9	2D	2600	LDA	#'- MINUS SIGN
0925-	8D	2D	2610	STA	SIGN.CHAR
			2620		*****
0928-	38		2630	SEC	
0929-	AD	36	2640	LDA	DAC.EXPONENT
092C-	E9	40	2650	SBC	#\$40 REMOVE OFFSET
092E-	10	0A	2660	BPL	.4
0930-	49	FF	2670	EOR	#\$FF
0932-	8D	32	2680	STA	NO.LEADING.ZEROES
0935-	EE	32	2690	INC	NO.LEADING.ZEROES
0938-	A9	00	2700	LDA	#0
093A-	8D	33	2710	STA	NO.LEADING.DIGITS
			2720		*****
093D-	38		2730	SEC	
093E-	AD	33	2740	LDA	NO.LEADING.DIGITS
0941-	E9	12	2750	SBC	#18
0943-	30	08	2760	BMI	.5
0945-	8D	34	2770	STA	NO.INTEGRAL.ZEROES
0948-	A9	12	2780	LDA	#18 18 SIGNIF DIGITS MAX
094A-	8D	33	2790	STA	NO.LEADING.DIGITS
			2800		*****
094D-	18		2810	CLC	CALCULATE TOTAL # OF DIGITS
094E-	AD	2C	2820	LDA	SIGN.SIZE
0951-	6D	33	2830	ADC	NO.LEADING.DIGITS
0954-	6D	34	2840	ADC	NO.INTEGRAL.ZEROES
0957-	6D	2A	2850	ADC	D
095A-	69	01	2860	ADC	#1

095C-	8D	2F	08	2870	STA WW	
				2880	*-----	
095F-	38			2890	SEC	
0960-	AD	2A	08	2900	LDA D	
0963-	ED	32	08	2910	SBC NO.LEADING.ZEROS	
0966-	8D	30	08	2920	STA DD	
				2930	*-----	
0969-	38			2940	SEC	
096A-	AD	29	08	2950	LDA W	
096D-	ED	2F	08	2960	SBC WW	
0970-	30	59		2970	BMI .14	...OVERFLOW
0972-	8D	35	08	2980	STA NO.LEADING.BLANKS	
0975-	AD	33	08	2990	LDA NO.LEADING.DIGITS	
0978-	DO	08		3000	BNE .6	
097A-	CE	35	08	3010	DEC NO.LEADING.BLANKS	
097D-	10	03		3020	BPL .6	
097F-	EE	35	08	3030	INC NO.LEADING.BLANKS IT WENT -, MAKE 0	
				3040	*---STORE LEADING BLANKS---	
0982-	A9	20		3050	.6 LDA #' ' BLANK	
0984-	AC	35	08	3060	LDY NO.LEADING.BLANKS	
0987-	20	FA	09	3070	JSR STORE.N.CHARS	
				3080	*---STORE SIGN---	
098A-	AD	2D	08	3090	LDA SIGN.CHAR	
098D-	FO	03		3100	BEQ .8	
098F-	20	01	0A	3110	JSR STORE.CHAR	
				3120	*---STORE INTEGRAL DIGITS---	
0992-	AC	33	08	3130	.8 LDY NO.LEADING.DIGITS	
0995-	FO	05		3140	BEQ .10	
0997-	20	F3	09	3150	JSR STORE.N.DIGITS	
099A-	FO	06		3160	BEQ .11	...ALWAYS
099C-	AD	2E	08	3170	.10 LDA ZERO.CHAR NO INTEGER PART,SO PRINT 0	
099F-	20	01	0A	3180	JSR STORE.CHAR	
09A2-	A9	30		3190	.11 LDA #'0	
09A4-	AC	34	08	3200	LDY NO.INTEGRAL.ZEROS	
09A7-	20	FA	09	3210	JSR STORE.N.CHARS	
				3220	*---STORE FRACTION---	
09AA-	A9	2E		3230	LDA #'.	
09AC-	20	01	0A	3240	JSR STORE.CHAR	
09AF-	AD	30	08	3250	LDA DD	
09B2-	OD	32	08	3260	ORA NO.LEADING.ZEROS	
09B5-	FO	0F		3270	BEQ .13	
09B7-	AD	2E	08	3280	LDA ZERO.CHAR	
09BA-	AC	32	08	3290	LDY NO.LEADING.ZEROS	
09BD-	20	FA	09	3300	JSR STORE.N.CHARS	
09C0-	AC	30	08	3310	LDY DD	
09C3-	20	F3	09	3320	JSR STORE.N.DIGITS	
				3330	*---TERMINATE STRING---	
09C6-	A9	00		3340	.13 LDA #0	
09C8-	4C	01	0A	3350	JMP STORE.CHAR	
				3360	*-----	
09CB-	A9	2A		3370	.14 LDA #' ' FILL FIELD WITH STARS	
09CD-	AC	29	08	3380	LDY W	
09D0-	20	FA	09	3390	JSR STORE.N.CHARS	
09D3-	4C	C6	09	3400	JMP .13	
				3410	*-----	
				3420	* STORE NEXT (Y) DIGITS	
				3430	*-----	
09D6-	AD	31	08	3440	SND..1 LDA DIGIT.PICKER	
09D9-	C9	14		3450	CMP #20	
09DB-	90	0A		3460	BCC .1	
09DD-	A9	00		3470	LDA #0	
09DF-	FO	0E		3480	BEQ .2	...ALWAYS
09E1-	4A			3490	.1 LSR	LEFT/RIGHT --> C
09E2-	AA			3500	TAX	INDEX --> X
09E3-	EE	31	08	3510	INC DIGIT.PICKER	
09E6-	BD	37	08	3520	LDA DAC.HI,X	
09E9-	BO	04		3530	BCS .2	
09EB-	4A			3540	LSR	
09EC-	4A			3550	LSR	
09ED-	4A			3560	LSR	
09EE-	4A			3570	LSR	
09EF-	20	FD	09	3580	.2 JSR STORE.DIGIT	
09F2-	88			3590	DEY	
				3600	STORE.N.DIGITS	
09F3-	DO	E1		3610	BNE SND..1	
09F5-	60			3620	RTS	
				3630	*-----	

Apple Peripherals Are All We Make

That's Why We're So Good At It!



THE NEW TIMEMASTER II H.O.

- Absolutely, positively, totally PRO-DOS and DOS 3.3 compatible.
- Time in hours, minutes, seconds and milliseconds (the ONLY PRO-DOS compatible card with millisecond capability).
- 24 hour military format or 12 hour with AM/PM format.

- Date with year, month, day of week and leap year.
- The easiest programming in BASIC.
- Eight software controlled interrupts so you can run two programs at the same time (many examples are included).
- Compatible with ALL of Apple's languages. Many sample programs for machine code, Applesoft, CP/M and Pascal on 2 disks.
- On-board timer lets you time any interval up to 48 days long down to the nearest millisecond.
- Rechargeable nickel-cadmium battery will last over 20 years.
- Two BSR/serial ports for future expansion.

Full emulation of all other clocks. Yes, we emulate Brand A, Brand T, Brand P, Brand C, Brand S and Brand M too. It's easy for the H.O. to emulate other clocks, we just drop off features. That's why the H.O. can emulate others, but none of the others emulate us.

The Timemaster II H.O. will automatically emulate the correct clock card for the software you're using. You can also give the H.O. a simple command to tell it which clock to emulate. This is great for writing programs for those poor unfortunates that bought some other clock card.

Of course, most programs will use the Timemaster II H.O. in its native mode, but its comforting to know that you can use programs written for other products without any modification.

REMOTE CONTROL

Our BSR X-10 interface option for the H.O. allows you to remotely control up to 16 lights and electrical appliances through your BSR X-10 home control system in your home or office. You're already wired because a BSR system sends its signals over regular 120 volt wiring. That means you can control any electrical device in your home or office without any additional wiring.

PRICE \$129.00 BSR Option \$49.00

MemoryMaster IIe 128K RAM Card

- Expands your Apple IIe to 192K memory.
- Provides an 80 column text display.
- Compatible with all Apple IIe 80 column and extended 80 column card software (same physical size as Apple's 64K card).
- Can be used as a solid state disk drive to make your programs run up to 20 times FASTER (the 64K configuration will act as half a drive).
- Permits your IIe to use the new double high resolution graphics.
- Automatically expands Visicalc to 95K storage in 80 columns! The 64K config. is all that's needed, 128K can take you even higher.
- PRO-DOS will use the MemoryMaster IIe as a high speed disk drive.
- The 64K MemoryMaster IIe will automatically expand Apple Works to 55K storage. The 128K MemoryMaster IIe will expand Apple Works to 101K storage.
- High Speed disk emulation for BASIC, Pascal and CP/M is available at a very low cost. NOT copy protected.
- Documentation included, we show you how to use all 192K.

If you already have Apple's 64K card, just order the MEMORYMASTER IIe with 64K and use the 64K from your old board to give you a full 128K. (The board is fully socketed so you simply plug in more chips.)

MemoryMaster IIe with 128K \$249
Upgradable MemoryMaster IIe with 64K \$169
Non-Upgradable MemoryMaster IIe with 64K \$149

Z-80 PLUS



- TOTALLY compatible with ALL CP/M software.
- The only Z-80 card with a special 2K "CP/M detector" chip.
- Fully compatible with microsoft disks (no pre-boost required).
- Specifically designed for the Apple IIe (runs just as fast in the II+ and Franklin).

- Runs WORD STAR, dBASE II, COBOL-80, FORTRAN-80, PEACHTREE and ALL other CP/M software with no pre-boost.
- A semi-count I.C. and low parts count allows the Z-80 Plus to fly thru CP/M programs at a very low power level. (We use the Z-80A at fast 4MHZ.)
- Does EVERYTHING the other Z-80 boards do, plus Z-80 interrupts. Don't confuse the Z-80 Plus with crude copies of the microsoft card. The Z-80 Plus employs a much more sophisticated and reliable design. With the Z-80 Plus you can access the largest body of software in existence. Two computers in one and the advantages of both, all at an unbelievably low price.

PRICE \$139.00

VIEWMASTER 80

There used to be about a dozen 80 column cards for the Apple, now there's only ONE.

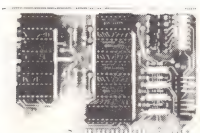
- TOTALLY Videx Compatible.
- 80 characters by 24 lines, with a sharp 7x9 dot matrix.
- On-board 40/80 soft video switch with manual 40 column override.
- Fully compatible with ALL Apple languages and software—there are NO exceptions.
- Low power consumption through the use of CMOS devices.
- All connections are made with standard video connectors.
- Both upper and lower case characters are standard.
- All new design (using a new Microprocessor based C.R.T. controller) for a beautiful razor sharp display.
- The VIEWMASTER incorporates all the features of all other 80 column cards, plus many new improvements.

	PRICE	80 COL IN SOFTWARE	80 COL BY HARDWARE	LOW POWER DESIGN	NO LUTTERING	THE BEST AUDIO	LIGHT PEN INPUT	80 COLUMN OVERMODE	INVERSE CHARACTER
VIEWMASTER	159	YES	YES	YES	YES	YES	YES	YES	YES
SLIPSTREAM	MORE	NO	YES	NO	NO	NO	NO	YES	YES
WIZARD80	MORE	NO	NO	NO	NO	YES	NO	YES	YES
VISION80	MORE	YES	YES	NO	NO	YES	NO	NO	NO
OMNIVISION	MORE	NO	YES	NO	NO	NO	NO	YES	YES
VIEWMASTER	MORE	YES	YES	NO	NO	YES	NO	NO	YES
SMARTER	MORE	YES	YES	NO	NO	NO	YES	YES	NO
VIDEOTERM	MORE	NO	YES	YES	NO	YES	YES	NO	YES

The VIEWMASTER 80 works with all 80 column applications including CP/M, Pascal, WordStar, Format II, Easywriter, Apple Writer II, VisiCalc, and all others. The VIEWMASTER 80 is THE MOST complete 80 column card you can buy at ANY price.

Thousands SOLD at \$179 NOW ONLY \$159.00

SUPER MUSIC SYNTHESIZER - END MOCKINGBOREDOM



- Complete 16 voice music synthesizer on one card. Just plug it into your Apple, connect the audio cable (supplied) to your stereo, boot the disk supplied and you are ready to input and play songs.
- It's easy to program music with our compose software. You will start right away at inputting your favorite songs. The Hi-Res screen shows what you have entered in standard sheet music format.

- Now with new improved software for the easiest and the fastest music input system available anywhere.
- We give you lots of software. In addition to Compose and Play programs, 2 disks are filled with over 30 songs ready to play.
- Easy to program in Basic to generate complex sound effects. Now your games can have explosions, phaser zaps, train whistles, death cries. You name it, this card can do it.
- Four white noise generators which are great for sound effects.
- Plays music in true stereo as well as true discrete quadraphonic.
- Full control of attack, volume, decay, sustain and release.
- Our card will play notes from 30HZ to beyond human hearing.
- Automatic shutoff on power-up or if reset is pushed.
- Many many more features.

PRICE \$159.00

Our boards are far superior to most of the consumer electronics made today. All I.C.'s are in high quality sockets with mil-spec. components used throughout. P.C. boards are glass-epoxy with gold contacts. Made in America to be the best in the world. All products work in the APPLE IIe, II+, and Franklin. The MemoryMaster IIe is IIe only. Applied Engineering also manufactures a full line of data acquisition and control products for the Apple. A/D converters and digital I/O cards, etc. Please call for more information. All our products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a no hassle **THREE YEAR WARRANTY**.

Send Check or Money Order to:

APPLIED ENGINEERING
P.O. Box 798
Carrollton, TX 75006

Call (214) 492-2027

8 a.m. to 11 p.m. 7 days a week
MasterCard, Visa & C.O.D. Welcome
No extra charge for credit cards

Texas Residents Add 5% Sales Tax
Add \$10.00 II Outside U.S.A.

Enable/Disable IRQ from Applesoft.....Bob Sander-Cederlof

If you have applied the patches to DOS 3.3 that we published in the January 1984 issue (pages 10,11), and if you now are using interrupts from such sources as the Timemaster II or a handy pushbutton, you have probably run into the need to enable and disable IRQ from within an Applesoft program. (That sentence is the kind you have to read without interruption, so I really should have begun the paragraph with SEI and ended it with CLI.)

What is need is four bytes of assembly language, at a location that you can CALL. For example:

```
300- 58    CLI
301- 60    RTS
302- 78    SEI
303- 60    RTS
```

If those four bytes are in memory as shown, you can CALL 768 to enable IRQ interrupts, and CALL 771 to disable them. You can install the four bytes like this:

```
100 POKE 768,88: POKE 769,96
110 POKE 770,120:POKE 771,96
```

Now there are often times when poking into page 3 is not possible. Are there other tricky ways to get those bytes installed, without using page 3?

I found a half dozen or so. First, realize that the four bytes only need to be there when you call them. The rest of the time the same locations could be used for other purposes. For example, we could poke them into the input buffer at \$200, as long as we do it every time we CALL it:

```
100 POKE 512,88:POKE 513,96:CALL 512
    to enable interrupts, or
```

```
500 POKE 512,120:POKE 513,96:CALL 512
    to disable them.
```

The result of a multiplication or division is left, sometimes normalized and sometimes not, in \$62...\$66. If we find two numbers whose product leaves the bytes \$58 and \$60 at \$62 and \$63, we could CALL 98:

```
100 X = 1*707 : CALL 98 : REM ENABLE IRQ
200 X = 1*963 : CALL 98 : REM DISABLE IRQ
```

On the next page is a table showing the various methods I found:

Enable (CLI..RTS)

Disable (SEI..RTS)

100 POKE 38,88

100 POKE 38,120

110 POKE 39,96

110 POKE 39,96

120 CALL 38

120 CALL 38

100 CALL 8411232-8411065

100 CALL 8419424-8419257

100 GOSUB 24664

100 GOSUB 24696

24664 CALL 117:RETURN

24696 CALL 117:RETURN

100 X = 1*707 : CALL 98

100 X = 1*963 : CALL 98

100 X = RND(-8411323.5)

100 X = RND(-8419424.5)

110 CALL 203

110 CALL 203

100 HOME:FLASH:PRINT "X "

100 HOME:FLASH:PRINT "8 "

110 NORMAL:CALL1024

110 NORMAL:CALL1024

Can you figure out how all these work? They are pretty tricky!
Can you think of some more?

Now you can monitor and control the world (or at least your part of it) with a little help from

APPLIED ENGINEERING

12 BIT, 16 CHANNEL, PROGRAMMABLE GAIN A/D

- All new 1984 design incorporates the latest in state-of-art I.C. technologies.
- Complete 12 bit A/D converter, with an accuracy of 0.02%!
- 16 single ended channels (single ended means that your signals are measured against the Apple's GND.) or 8 differential channels. Most all the signals you will measure are single ended.
- 9 software programmable full scale ranges, any of the 16 channels can have any range at any time. Under program control, you can select any of the following ranges: ± 10 volts, $\pm 5V$, $\pm 2.5V$, $\pm 1.0V$, $\pm 500mV$, $\pm 250mV$, $\pm 100mV$, $\pm 50mV$, or $\pm 25mV$.
- Very fast conversion (25 micro seconds).
- Analog input resistance greater than 1,000,000 ohms.
- Laser-trimmed scaling resistors.
- Low power consumption through the use of CMOS devices.
- The user connector has +12 and -12 volts on it so you can power your sensors.
- Only elementary programming is required to use the A/D.
- The entire system is on one standard size plug in card that fits neatly inside the Apple.
- System includes sample programs on disk.

PRICE \$319

A few applications may include the monitoring of ● flow ● temperature ● humidity ● wind speed ● wind direction ● light intensity ● pressure ● RPM ● soil moisture and many more.

8 BIT, 8 CHANNEL A/D

- 8 Channels
 - 8 Bit Resolution
 - On Board Memory
 - Fast Conversion (0.78 ms per channel)
 - A/D Process Totally Transparent to Apple (looks like memory)
- The APPLIED ENGINEERING A/D BOARD is an 8 bit, 8 channel memory buffered, data acquisition system. It consists of an 8 bit A/D converter, an 8 channel multiplexer and 8 x 8 random access memory.
- The analog to digital conversion takes place on a continuous, channel sequencing basis. Data is automatically transferred to on board memory at the end of each conversion. No A/D converter could be easier to use.
- Our A/D board comes standard with 0, 10V full scale inputs. These inputs can be changed by the user to 0, -10V, or -5V, +5V or other ranges as needed.
- The user connector has +12 and -12 volts on it so you can power your sensors.
- Accuracy: 0.3%
 - Input Resistance: 20K Ohms Typ

PRICE \$129.00

SIGNAL CONDITIONER

Our 8 channel signal conditioner is designed for use with both our A/D converters. This board incorporates 8 I.E.T. op-amps, which allow almost any gain or offset, for example: an input signal that varies from 2.00 to 2.15 volts or a signal that varies from 0 to 50 mV can easily be converted to 0-10V output for the A/D.

The signal conditioner's outputs are a high quality 16 pin gold I.C. socket that matches the one on the A/D's so a simple ribbon cable connects the two. The signal conditioner can be powered by your Apple or from an external supply.

FEATURES

- 4.5" square for standard card cage and 4 mounting holes for standard mounting. The signal conditioner does not plug into the Apple, it can be located up to 1/2 mile away from the A/D.
- 22 pin, 156 spacing edge card input connector (extra connectors are easily available i.e. Radio Shack).
- Large bread board area.
- Full detailed schematic included.

PRICE \$79.00

DIGITAL INPUT/OUTPUT BOARD

- Provides 8 buffered outputs to a standard 16 pin socket for standard dip ribbon cable connection.
- Power-up reset assures that all outputs are off when your Apple is turned on.
- Features 8 inputs that can be driven from TTL logic or any 5 volt source.
- Your inputs can be anything from high speed logic to simple switches.
- Very simple to program, just PEEK at the data.
- Now, on one card, you can have 8 digital outputs and 8 digital inputs each with its own connector. The super input/output board is your best choice for any control application.

The SUPER INPUT/OUTPUT board manual includes many programs for inputs and outputs. A detailed schematic is included.

Some applications include:

Burglar alarm, direction sensing, use with relays to turn on lights, sound buzzers, start motors, control tape recorders and printers, use with digital joystick.

PRICE \$69.00

Please see our other full page ad in this magazine for information on Applied Engineering's Timemaster Clock Card and other products for the Apple.

Our boards are far superior to most of the consumer electronics made today. All I.C.'s are in high quality sockets with mil-spec. components used throughout. P.C. boards are glass-epoxy with gold contacts. Made in America to be the best in the world. All products compatible with Apple II and IIe.

Applied Engineering's products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a no hassle three year warranty.

Texas Residents Add 5% Sales Tax
Add \$10.00 if Outside U.S.A.

Send Check or Money Order to:
APPLIED ENGINEERING
P.O. Box 798
Carrollton, TX 75006

Call (214) 492-2027
7 a.m. to 11 p.m. 7 days a week
MasterCard, Visa & C.O.D. Welcome
No extra charge for credit cards

Line Number Cross Reference.....Bill Morgan

Have you ever had to modify a BASIC program written by someone who didn't seem to know what he was doing? Deciphering several hundred undocumented lines of split FOR/NEXTs and tangled GOTOs can lead to a severe headache. We recently had a consulting job that involved just such a project: one program to be altered was about a hundred sectors of spaghetti-plate Applesoft. A couple of the biggest problems were figuring out which lines used a particular variable, and what lines called others, or were called from where.

Back in November of 1980, AAL published a Variable Cross Reference program which neatly took care of the first problem by producing a listing in alphabetical order of all the variables used and all the lines using them. At the end of that article, Bob S-C pointed out that the program could, with some effort, be modified into just the sort of Line Number Cross Reference we now needed. Well, I drew the job of making that modification, and here's what I came up with.

The Basis

These Cross Reference programs use a hash-chain data structure to store the called and calling line numbers. Each called line has its own list of lines which refer to it. We locate these lists by using the upper six bits of the line number for an index into a table located at \$280. This table contains the address of the beginning of each of the 64 possible chains. Each chain is made up of the data for a range of 1024 possible called line numbers. The first one has called lines 0-1023, the second has 1024-2047, and so on.

The entry for each called line is made up of a pointer to the next called line in that chain, this called line number, a pointer to the next calling line, and the number of this calling line. Each subsequent calling line entry has only the last four bytes. A pointer with a value of zero marks the end of each chain and each list.

VCR used three characters for each variable: the first two letters of the variable name and a type designator of "\$", "%", or " ". The first character was the hash index and the last two characters were stored at the beginning of each variable's chain. LCR uses the high-order 6 bits of the called line number for the hash index and stores both bytes of the number in the chain. This is slightly redundant, so if you want to store more information about the called line, you can use the upper six bits of the chain entry.

VCR stored the calling line numbers with the high byte first, backwards from usual 6502 practice. This was done so the same search-compare code could handle both variable names and line numbers. To simplify the conversion I kept the same structure, even though it's no longer strictly necessary.

The Program

LCR, the overall control level, is identical to VCR and just calls the other routines.

INITIALIZATION prepares a couple of pointers and zeroes the hash table. The only difference here is the size of the hash table.

PROCESS.LINE is also the same as in VCR. This routine steps through the lines of the Applesoft program, moving the calling line number into our data area and JSRING to SCAN.FOR.CALLS to work on each line.

SCAN.FOR.CALLS is the first really new section of code. We start by setting a flag used to mark ON ... GO statements. Then we step through the bytes of the line, looking for tokens that call another line. GOTO and GOSUB are processed immediately. For a THEN token we check to see if the next character is a number. If it is, we deal with it; if not, we go on. If we find an ON token, we set the flag and keep looking. After a GOTO or GOSUB we check ONFLAG. If there was an ON, we look for a comma to mark another called line number.

PROCESS.CALL first converts the ASCII line number of the called line into a two-byte binary number and then searches the data structure for that line. If it is there, we simply add this calling line to the list. If we don't find the called line we create a new entry for it.

CONVERT.LINE.NUMBER is lifted straight from Applesoft's LINGET, at \$DA0C.

NEXT.CHAR is a utility routine to get the next byte from the program and advance the pointer.

SEARCH.CALL.TABLE starts the search pointer on the appropriate chain.

CHAIN.SEARCH uses the pointer in an entry to step to the next entry. If the pointer is zero, then there is no next entry and the search has failed. We then compare the line number in the entry to the one we're looking for. If the entry is less than the search key, we go on. If it is equal, we update the pointer and report success. If we hit an entry greater than the key, the search fails and we return.

SEARCH.LINE.CHAIN is called after SEARCH.CALL.TABLE has found a match. Here we move the pointer to the calling line field of the matching entry and use the current calling line for a search key.

ADD.NEW.ENTRY first updates the pointers in the previous entry and this new entry, and the end-of-table pointer. We then make sure there is room for the new entry and move the data up into the new space.


```

3640 *   STORE (Y) OF THE CHARACTER IN (A)
3650 *   (Z-STATUS IF COUNT IS 0)
3660 *-----
09F6- 20 01 0A 3670 SNC..1 JSR STORE.CHAR
09F9- 88        3680 DEY
3690 STORE.N.CHARS
09FA- D0 FA 3700 BNE SNC..1
09FC- 60      3710 RTS
3720 *-----
3730 *   STORE A CHAR IN THE BUFFER
3740 *-----
3750 STORE.DIGIT
09FD- 29 0F 3760 AND #$0F
09FF- 09 30 3770 ORA #'0'
3780 STORE.CHAR
0A01- AE 2B 08 3790 LDX INDEX
0A04- 9D 00 08 3800 STA FOUT.BUF,X
0A07- EE 2B 08 3810 INC INDEX
0A0A- 60      3820 RTS
3830 *-----
3840 *   ROUND DAC TO (D) DECIMAL PLACES
3850 *-----
3860 ROUND.DAC.D
0A0B- AD 41 08 3870 LDA DAC.SIGN GET THE SIGN
0A0E- 48      3880 PHA SAVE IT
0A0F- A9 28 3890 LDA #CON.1HALF
0A11- A0 0A 3900 LDY /CON.1HALF
0A13- 20 FF FF 3910 JSR MOVE.YA.ARG MOVE .5*10^-D INTO ARG
0A16- 68      3920 PLA GET SIGN
0A17- 8D 4D 08 3930 STA ARG.SIGN
0A1A- AD 2A 08 3940 LDA D GET # OF PLACES
0A1D- 49 FF 3950 EOR #$FF MAKE IT NEGATIVE BY 2S COMPLEMENT
0A1F- 38      3960 SEC ADD 1 DURING NEXT ADD
0A20- 69 40 3970 ADC #$40 ADD OFFSET
0A22- 8D 42 08 3980 STA ARG.EXPONENT
0A25- 4C FF FF 3990 JMP DADD ADD .5*10^-D;FOUT WILL TRUNCATE IT
4000 *-----
0A28- 40 50 00
0A2B- 00 00 00
0A2E- 00 00 00
0A31- 00 00 4010 CON.1HALF .HS 40500000000000000000

```

27 Enhancements for the S-C Macro Assembler 1.1 (Videx \$1000 ver.)

A 1723-byte patch for Videx 80-column board users providing On-Line editing with version 1.1 - All the edit commands are now available during normal input (non-edit mode) i.e. insert, delete, skip-to-character etc. The New/Extended Commands are:

Non-Edit Mode: Ctrl-A,B,C,D,F,x,K,L,M,N,P,Q,R,S,T,V,W,Z,@,
Ctrl-shift-N, Esc-C, PAG, USR, AUTO, VAL, MEM.
Edit Mode: Ctrl-K, P, Z.

The RETURN-key accepts the whole line irrespective of cursor position. USR toggles line-numbers on/off during listing and assembly. VAL now displays hex, binary and decimal results. Ctrl-K skips to start of comment column from any position. Ctrl-Z does an UPPER/lower-case toggle. Ctrl-W toggles between AUTO and MANUAL modes. After a CTRL-shift-N the next key pressed becomes the clear-to-tab character. Esc-C does a CATALOG. MEM shows free space in RAM and on Disk. (DOS 3.3) Ctrl-C continues editing from current linenumber. Ctrl-V allows the user to view source from current linenumber. Leading zeros are removed from linenumbers and page numbers if using .TI. The SYM routine is modified to exclude the clutter of local labels. Ctrl-P outputs the source separator "-----etc" in both the edit and non-edit modes. You can now use Ver 1.1 as a useful if limited 80-column line-oriented wordprocessor for your letters and reports. User RAM \$3C00-\$9600 (Maxfiles 3). All for \$15 - which buys a disk with Article, binary patch, installer and 980-line source code.

R.F.O'Brien,14 CLonshaugh Lawn, Coolock, Dublin 5, Ireland.

Now we are done with the routines devoted to building the Cross Reference tables. Interestingly, SEARCH.CALL.TABLE, CHAIN.SEARCH, SEARCH.LINE.CHAIN, and ADD.NEW.ENTRY are the real heart of this program, and the only change I had to make in these routines from VCR to LCR was to alter the method of figuring the hash index in SEARCH.CALL.TABLE. Next we come to getting the data back out of the tables and onto a display.

PRINT.REPORT first sets a pointer we'll be using later on and then steps through the hash table, calling PRINT.CHAIN for each entry found.

PRINT.CHAIN starts out by checking for a pause or abort signal from the keyboard. It then moves the current called line number into LINNUM, checks to see if it really exists, and prints it, followed by an asterisk if it is undefined. Now we move a pointer up to the start of the calling line list and call PRINT.LINNUM.CHAIN to display all the entries. The last step is to move the pointer up to the next called line in this chain, if any, and go back to do that one.

CHECK.DEFINITION keeps its own pointer into the program and steps along checking each called line to see if it actually exists. It provides a space or an asterisk to be printed after the line number.

PRINT.LINNUM.CHAIN displays the calling lines stored for each called line. We first tab to the next column (or line if necessary), then get the line number out of the list and print it. Lastly, we move the pointer up to the next entry, if any, and loop back.

TAB.NEXT.COLUMN prints enough blanks to move over to the next output position. If a new line is necessary, it checks the line number to see if the new line should go to the screen only, or also to a printer. This is Louis Pitz's addition, designed to automatically handle either 40- or 80-column output.

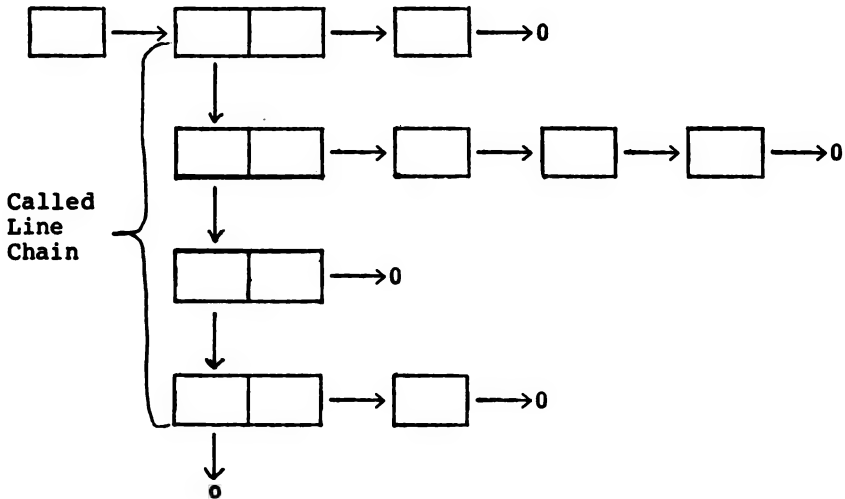
PRINT.LINE.NUMBER and CHECK.FOR.PAUSE are pretty standard routines to convert a two-byte binary number into five decimal characters, and to provide for pause/abort during display.

Well, now we have a Line Number Cross Reference to go along with the Variable Cross Reference. Now all that remains is to integrate the two programs into one master Applesoft Cross Reference Utility. Maybe you could call it with "&V" for VCR, or "&L" for LCR, and simply "&" to get both listings. Any takers out there?

PS: Bob suggested that I add a diagram of the hash chain structure, and a summary of the search process. OK, here they are...

One of
64 hash
pointers

Calling Line Lists



Found a Call

- * Use high 6 bits of called line number to index Hash Table
- * Get pointer from Hash Table to find start of chain
 - * If no pointer in Hash Table, make new entry
- * Search chain for same line number
 - * If not found, make new link in chain
 - * If found, search calling line list
- * Enter new calling line in list

```

1000 *SAVE S.LCR
1010 *-----
1020 *   LINE NUMBER CROSS REFERENCE
1030 *   FOR APPLESOFT PROGRAMS
1040 *
1050 *   Based on Variable Cross Reference
1060 *   Original by Bob S-C 11/80
1070 *   Modified by Louis Pitz 8/83
1080 *   Adapted by Bill Morgan 8/84
1090 *-----
1100   .OR $9000
1110   .TF B.LCR
1120 *-----
9000- A9 4C 1130   LDA #$4C           set & vector
9002- 8D F5 03 1140   STA $3F5
9005- A9 10 1150   LDA #LCR
9007- 8D F6 03 1160   STA $3F6
900A- A9 90 1170   LDA #LCR
900C- 8D F7 03 1180   STA $3F7
900F- 60 1190   RTS
1200 *-----
15- 1210 TEMP   .EQ $15
16- 1220 COUNTER .EQ $16
17- 1230 ONFLAG .EQ $17           ON ... GO flag
17- 1240 DEFFLAG .EQ $17
18- 1250 PNTR   .EQ $18,19       pointer into program
1A- 1260 LZFLAG .EQ $1A         leading zero flag

```

```

1A-      1270 DATA      .EQ $1A thru $1D
1A-      1280 NEXTLN    .EQ $1A,1B      address of next line
1C-      1290 LINNUM    .EQ $1C,1D      current line number
1E-      1300 STPNTR    .EQ $1E,1F      pointer into call table
9B-      1310 TPTR      .EQ $9B,9C      temp pointer
9D-      1320 ENTRY     .EQ $9D thru $A4 8 bytes
9F-      1330 CALL      .EQ ENTRY+2
A5-      1340 SIZE      .EQ $A5,A6
0280-    1350 HSHTBL     .EQ $280
          1360 #-----
67-      1370 PRGBOT     .EQ $67,68      beginning of program
69-      1380 LOMEM      .EQ $69,6A      beginning of variable space
6B-      1390 EOT        .EQ $6B,6C      end of variable table
          1400 #-----
2C-      1410 COMMA      .EQ '
8D-      1420 CR         .EQ $8D
AB-      1430 TKN.GOTO    .EQ $AB
B0-      1440 TKN.GOSUB   .EQ $B0
B4-      1450 TKN.ON      .EQ $B4
C4-      1460 TKN.THEN    .EQ $C4
          1470 #-----
24-      1480 MON.CH      .EQ $24
C000-    1490 KEYBOARD    .EQ $C000
C010-    1500 STROBE      .EQ $C010
D410-    1510 AS.MEMFULL  .EQ $D410
F94A-    1520 MON.PREL2   .EQ $F94A
FD8E-    1530 MON.CROUT   .EQ $FD8E
FDED-    1540 MON.COUT    .EQ $FDED
FDF0-    1550 MON.COUT1   .EQ $FDF0
          1560 #-----
9010- 20 23 90 1570 LCR      JSR INITIALIZATION
9013- 20 3E 90 1580 .1      JSR PROCESS.LINE
9016- D0 FB 1590 BNE .1      until end of program
9018- 20 B5 91 1600 JSR PRINT.REPORT
901B- 20 23 90 1610 JSR INITIALIZATION  erase call table
901E- A9 00 1620 LDA #0      clear $A4 so Applesoft
9020- 85 A4 1630 STA $A4      will work correctly
9022- 60 1640 RTS
          1650 #-----
          1660 INITIALIZATION
9023- A5 69 1670 LDA LOMEM      start call table
9025- 85 6B 1680 STA EOT        after program
9027- A5 6A 1690 LDA LOMEM+1
9029- 85 6C 1700 STA EOT+1
902B- A2 80 1710 LDX #80      # of bytes for hash pointers
902D- A9 00 1720 LDA #0
902F- 9D 7F 02 1730 .1 STA HSHTBL-1,X
9032- CA 1740 DEX
9033- D0 FA 1750 BNE .1
9035- A5 67 1760 LDA PRGBOT      start pointer at
9037- 85 18 1770 STA PNTR        beginning of program
9039- A5 68 1780 LDA PRGBOT+1
903B- 85 19 1790 STA PNTR+1
903D- 60 1800 RTS
          1810 #-----
          1820 PROCESS.LINE
903E- A0 03 1830 LDY #3      capture pointer and line #
9040- B1 18 1840 .1 LDA (PNTR),Y
9042- 80 1A 00 1850 STA DATA,Y
9045- 88 1860 DEY
9046- 10 F8 1870 BPL .1
9048- A5 1B 1880 LDA DATA+1      check if end
904A- F0 16 1890 BEQ .3      yes, return .EQ.
904C- 18 1900 CLC
904D- A5 18 1910 LDA PNTR
904F- 69 04 1920 ADC #4      adjust pointer to
9051- 85 18 1930 STA PNTR        skip over data
9053- 90 02 1940 EBC .2
9055- E6 19 1950 INC PNTR+1
9057- 20 63 90 1960 .2 JSR SCAN.FOR.CALLS
905A- A5 1A 1970 LDA DATA      point to next line
905C- 85 18 1980 STA PNTR
905E- A5 1B 1990 LDA DATA+1      and return .NE.
9060- 85 19 2000 STA PNTR+1
9062- 60 2010 .3 RTS
          2020 #-----

```

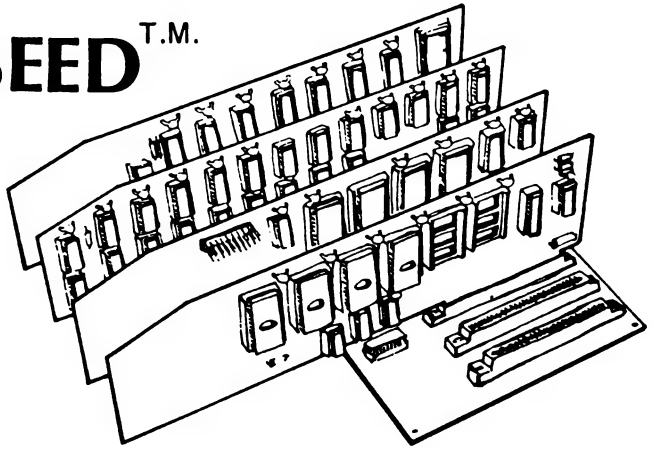
```

2030 SCAN.FOR.CALLS
9063- A9 FF 2040 LDA #$FF
9065- 85 17 2050 STA ONFLAG
9067- 20 07 91 2060 .1 JSR NEXT.CHAR
906A- F0 32 2070 BEQ .4
906C- C9 C4 2080 CMP #TKN.THEN scan for call token
906E- F0 10 2090 BEQ .2
9070- C9 AB 2100 CMP #TKN.GOTO
9072- F0 18 2110 BEQ .3
9074- C9 B0 2120 CMP #TKN.GOSUB
9076- F0 14 2130 BEQ .3
9078- C9 B4 2140 CMP #TKN.ON
907A- D0 EB 2150 BNE .1 no match, keep going
907C- 46 17 2160 LSR ONFLAG set flag for ON token
907E- 10 E7 2170 BPL .1 ...always
2180
9080- A0 00 2190 .2 LDY #0 after THEN, check
9082- B1 18 2200 LDA (PNTR),Y for line number
9084- C9 30 2210 CMP #'0
9086- 90 DF 2220 BCC .1 <0 isn't
9088- C9 3A 2230 CMP #'9+1
908A- B0 DB 2240 BCS .1 neither is >9
2250
908C- 20 9F 90 2260 .3 JSR PROCESS.CALL handle this call
908F- A5 17 2270 LDA ONFLAG are we in ON?
9091- 30 D0 2280 BMI SCAN.FOR.CALLS no, go on
9093- 20 07 91 2290 JSR NEXT.CHAR yes, look for comma
9096- F0 06 2300 BEQ .4 EOL
9098- C9 2C 2310 CMP #COMMA
909A- F0 F0 2320 BEQ .3 comma, get another call
909C- D0 C5 2330 BNE SCAN.FOR.CALLS ...always
2340
909E- 60 2350 .4 RTS
2360 -----
2370 PROCESS.CALL
909F- 20 C4 90 2380 JSR CONVERT.LINE.NUMBER
90A2- 20 14 91 2390 JSR SEARCH.CALL.TABLE
90A5- 90 13 2400 BCC .2 found same call
90A7- A9 00 2410 LDA #0
90A9- 85 A1 2420 STA ENTRY+4 start of line number chain
90AB- 85 A2 2430 STA ENTRY+5
90AD- A5 1D 2440 LDA LINNUM+1 MSB first
90AF- 85 A3 2450 STA ENTRY+6
90B1- A5 1C 2460 LDA LINNUM
90B3- 85 A4 2470 STA ENTRY+7
90B5- A9 08 2480 LDA #8 add 8 byte entry
90B7- 4C 74 91 2490 .1 JMP ADD.NEW.ENTRY
2500
90BA- 20 54 91 2510 .2 JSR SEARCH.LINE.CHAIN
90BD- 90 04 2520 BCC .3 found same line number
90BF- A9 04 2530 LDA #4 add 4 byte entry
90C1- D0 F4 2540 BNE .1 ...always
2550
90C3- 60 2560 .3 RTS
2570 -----
2580 CONVERT.LINE.NUMBER
90C4- A9 00 2590 LDA #0
90C6- 85 A0 2600 STA CALL+1
90C8- 85 9F 2610 STA CALL
90CA- 20 07 91 2620 .1 JSR NEXT.CHAR
90CD- F0 2F 2630 BEQ .2 EOL
90CF- 38 2640 SEC
90D0- E9 30 2650 SBC #'0 make value
90D2- 90 2A 2660 BCC .2 <0 isn't number
90D4- C9 0A 2670 CMP #9+1
90D6- B0 26 2680 BCS .2 >9 isn't number
90D8- 48 2690 PHA save value
90D9- A5 9F 2700 LDA CALL
90DB- 85 15 2710 STA TEMP
90DD- A5 A0 2720 LDA CALL+1 multiply CALL * 10
90DF- 0A 2730 ASL
90E0- 26 15 2740 ROL TEMP
90E2- 0A 2750 ASL
90E3- 26 15 2760 ROL TEMP
90E5- 65 A0 2770 ADC CALL+1
90E7- 85 A0 2780 STA CALL+1
90E9- A5 15 2790 LDA TEMP
90EB- 65 9F 2800 ADC CALL
90ED- 85 9F 2810 STA CALL

```

90EF-	06	AO	2820	ASL	CALL+1	
90F1-	26	9F	2830	ROL	CALL	
90F3-	68		2840	PLA		get value this digit
90F4-	65	AO	2850	ADC	CALL+1	and add it in
90F6-	85	AO	2860	STA	CALL+1	
90F8-	90	D0	2870	BCC	.1	
90FA-	E6	9F	2880	INC	CALL	
90FC-	B0	CC	2890	BCS	.1	...always
			2900			
90FE-	A5	18	2910	.2	LDA	PNTR
9100-	D0	02	2920		BNE	.3
9102-	C6	19	2930		DEC	PNTR+1
9104-	C6	18	2940	.3	DEC	PNTR
9106-	60		2950		RTS	
			2960			
			2970			
9107-	A0	00	2980		LDY	#0
9109-	B1	18	2990		LDA	(PNTR),Y
910B-	F0	06	3000		BEQ	.1
910D-	E6	18	3010		INC	PNTR
910F-	D0	02	3020		BNE	.1
9111-	E6	19	3030		INC	PNTR+1
9113-	60		3040	.1	RTS	
			3050			
			3060			
9114-	A5	9F	3070		LDA	CALL
9116-	29	FC	3080		AND	##FC
9118-	4A		3090		LSR	
9119-	69	80	3100		ADC	#HSHTBL
911B-	85	1E	3110		STA	STPNTR
911D-	A9	02	3120		LDA	/HSHTBL
911F-	69	00	3130		ADC	#0
9121-	85	1F	3140		STA	STPNTR+1
			3150			
			3160			
			3170			
9123-	A0	00	3180	.1	LDY	#0
9125-	B1	1E	3190		LDA	(STPNTR),Y
9127-	85	9B	3200		STA	TPTR
9129-	C8		3210		INY	
912A-	B1	1E	3220		LDA	(STPNTR),Y
912C-	F0	1A	3230		BEQ	.4
912E-	85	9C	3240		STA	TPTR+1
9130-	A2	02	3250		LDX	#2
9132-	A0	02	3260		LDY	#2
9134-	B1	9B	3270	.2	LDA	(TPTR),Y
9136-	D9	9D	3280		CMP	ENTRY,Y
9139-	90	08	3290		BCC	.3
913B-	D0	0B	3300		BNE	.4
913D-	CA		3310		DEX	
913E-	F0	0A	3320		BEQ	.5
9140-	C8		3330		INY	
9141-	D0	F1	3340		BNE	.2
			3350			
9143-	20	4A	3360	.3	JSR	.5
9146-	90	DB	3370		BCC	.1
			3380			
9148-	38		3390	.4	SEC	
9149-	60		3400		RTS	
			3410			
914A-	A5	9B	3420	.5	LDA	TPTR
914C-	85	1E	3430		STA	STPNTR
914E-	A5	9C	3440		LDA	TPTR+1
9150-	85	1F	3450		STA	STPNTR+1
9152-	18		3460		CLC	
9153-	60		3470		RTS	
			3480			
			3490			
9154-	18		3500			
9155-	A5	1E	3510		LDA	STPNTR
9157-	69	04	3520		ADC	#4
9159-	85	9D	3530		STA	ENTRY
915B-	A5	1F	3540		LDA	STPNTR+1
915D-	69	00	3550		ADC	#0
915F-	85	9E	3560		STA	ENTRY+1
9161-	A9	9D	3570		LDA	#ENTRY
9163-	85	1E	3580		STA	STPNTR
9165-	A9	00	3590		LDA	/ENTRY
9167-	85	1F	3600		STA	STPNTR+1

APPLESEED^{T.M.}



Appleseed is a complete computer system. It is designed using the bus conventions established by Apple Computer for the Apple II. Appleseed is designed as an alternative to using a full Apple II computer system. The Appleseed product line includes more than a dozen items including CPU, RAM, EPROM, UART, UNIVERSAL Boards as well as a number of other compatible items. This ad will highlight the Mother board.

BX-DE-12 MOTHER BOARD

The BX-DE-12 Mother board is designed to be fully compatible with all of the Apple conventions. Ten card slots are provided. Seven of the slots are numbered in conformance with Apple standards. The additional three slots, lettered A, B and C, are used for boards which don't require a specific slot number. The CPU, RAM and EPROM boards are often placed in the slots A, B and C.

The main emphasis of the Appleseed system is illustrated by the Mother Board. The absolute minimum amount of circuitry is placed on the Mother Board; only the four ICs which are required for card slot selection are on the mother board. This approach helps in packaging (flexibility & smaller size), cost (buy only what you need) and repairability (isolate and fix problems through board substitution).

Appleseed products are made for O.E.M.s and serious industrial/scientific users. Send for literature on the full line of Appleseed products; and, watch here, each month, for additional items in the Appleseed line.

Appleseed products are not sold through computer stores.

Order direct from our plant in California.

Apple is a registered trademark of Apple Computer, Inc.

DOUGLAS ELECTRONICS

718 Marina Blvd., San Leandro, CA 94577 • (415) 483-8770

```

9169- A5 1C 3610 LDA LINNUM          put line number into symbol
916B- 85 A0 3620 STA ENTRY+3
916D- A5 1D 3630 LDA LINNUM+1
916F- 85 9F 3640 STA ENTRY+2
9171- 4C 23 91 3650 JMP CHAIN.SEARCH
3660 -----
3670 ADD.NEW.ENTRY
9174- 85 A5 3680 STA SIZE
9176- 18 3690 CLC                      see if room
9177- A2 01 3700 LDY #1
9179- A0 00 3710 LDY #0
917B- 84 A6 3720 STY SIZE+1
917D- B1 1E 3730 .1 LDA (STPNTR),Y      get current pointer
917F- 99 9D 00 3740 STA ENTRY,Y      into new entry
9182- B9 6B 00 3750 LDA EOT,Y      point old entry
9185- 91 1E 3760 STA (STPNTR),Y    to this one
9187- 99 9B 00 3770 STA TPTR,Y
918A- 79 A5 00 3780 ADC SIZE,Y      and adjust end-of-table
918D- 99 6B 00 3790 STA EOT,Y
9190- C8 3800 INY
9191- CA 3810 DEX
9192- 10 E9 3820 BPL .1          now do low-bytes
3830 -----
3840 LDA EOT
9194- A5 6B 3850 CMP #LCR          see if there's going to be enough room
9196- C9 10 3860 LDA EOT+1
9198- A5 6C 3870 SBC /LCR
919A- E9 90 3880 BCS .3          MEM FULL error
919C- B0 14 3890 -----
3900 move entry into call table
919E- A4 A5 3900 LDY SIZE
91A0- 88 3910 DEY
91A1- B9 9D 00 3920 .2 LDA ENTRY,Y
91A4- 91 9B 3930 STA (TPTR),Y
91A6- 88 3940 DEY
91A7- 10 F8 3950 BPL .2
91A9- A5 9B 3960 LDA TPTR
91AB- 85 1E 3970 STA STPNTR
91AD- A5 9C 3980 LDA TPTR+1
91AF- 85 1F 3990 STA STPNTR+1
91B1- 60 4000 RTS
4010 -----
91B2- 4C 10 D4 4020 .3 JMP AS.MEMFULL    abort with error message
4030 -----
4040 PRINT.REPORT
91B5- A5 67 4050 LDA PRGBOT
91B7- 85 18 4060 STA PNTR          start defined line search
91B9- A5 68 4070 LDA PRGBOT+1      at beginning of program
91BB- 85 19 4080 STA PNTR+1
91BD- A9 00 4090 LDA #0          start at chain 0
91BF- 85 15 4100 .1 STA TEMP
91C1- 0A 4110 ASL
91C2- A8 4120 TAY
91C3- B9 81 02 4130 LDA HSHTL+1,Y
91C6- F0 0A 4140 BEQ .2          no entries for this chain
91C8- 85 1F 4150 STA STPNTR+1
91CA- B9 80 02 4160 LDA HSHTL,Y
91CD- 85 1E 4170 STA STPNTR
91CF- 20 DB 91 4180 JSR PRINT.CHAIN
91D2- E6 15 4190 .2 INC TEMP
91D4- A5 15 4200 LDA TEMP
91D6- C9 40 4210 CMP #40
91D8- 90 E5 4220 BCC .1          still more chains
91DA- 60 4230 RTS          finished
4240 -----
4250 PRINT.CHAIN
91DB- 20 DC 92 4260 JSR CHECK.FOR.PAUSE
91DE- F0 3A 4270 BEQ .1          <CR> abort
91E0- A0 02 4280 LDY #2
91E2- B1 1E 4290 LDA (STPNTR),Y
91E4- 85 1D 4300 STA LINNUM+1
91E6- C8 4310 INY
91E7- B1 1E 4320 LDA (STPNTR),Y
91E9- 85 1C 4330 STA LINNUM
91EB- 20 1D 92 4340 JSR CHECK.DEFINITION
91EE- 20 95 92 4350 JSR PRINT.LINE.NUMBER
91F1- A5 17 4360 LDA DEFFLAG      *** or *
91F3- 20 ED FD 4370 JSR MON.COUT

```


91F6-	18		4380	CLC	
91F7-	A5	1E	4390	LDA STPNTR	
91F9-	69	04	4400	ADC #4	point at line # chain
91FB-	85	9B	4410	STA TPTR	
91FD-	A5	1F	4420	LDA STPNTR+1	
91FF-	69	00	4430	ADC #0	
9201-	85	9C	4440	STA TPTR+1	
9203-	20	46	4450	JSR PRINT.LINNUM.CHAIN	
9206-	20	8E	4460	JSR MON.CROUT	
9209-	A0	01	4470	LDY #1	
920B-	B1	1E	4480	LDA (STPNTR),Y	pointer to next call
920D-	F0	0D	4490	BEQ .2	no more
920F-	48		4500	PHA	
9210-	88		4510	DEY	
9211-	B1	1E	4520	LDA (STPNTR),Y	
9213-	85	1E	4530	STA STPNTR	
9215-	68		4540	PLA	
9216-	85	1F	4550	STA STPNTR+1	
9218-	D0	00	4560	BNE .1	...always
			4570		
921A-	68		4580	.1 PLA	return to top level
921B-	68		4590	PLA	if <CR> abort
921C-	60		4600	.2 RTS	
			4610	-----	
			4620	CHECK.DEFINITION	
921D-	A0	03	4630	LDY #3	
921F-	A2	01	4640	LDX #1	
9221-	B1	18	4650	.1 LDA (PNTR),Y	look at next line in program
9223-	D5	1C	4660	CMP LINNUM,X	
9225-	90	0F	4670	BCC .4	< our number, get new line
9227-	D0	08	4680	BNE .2	> " " , not defined
9229-	88		4690	DEY	= " " , go on
922A-	CA		4700	DEX	now do low order bytes
922B-	10	F4	4710	BPL .1	
922D-	A9	A0	4720	LDA # " "	found it!
922F-	D0	02	4730	BNE .3	...always
			4740		
9231-	A9	AA	4750	.2 LDA #***	flag undefined line
9233-	85	17	4760	.3 STA DEFFLAG	
9235-	60		4770	RTS	
			4780		
9236-	A0	00	4790	.4 LDY #0	
9238-	B1	18	4800	LDA (PNTR),Y	lo-byte of next line address
923A-	48		4810	PHA	
923B-	C8		4820	INY	
923C-	B1	18	4830	LDA (PNTR),Y	and hi-byte
923E-	85	19	4840	STA PNTR+1	
9240-	68		4850	PLA	
9241-	85	18	4860	STA PNTR	
9243-	4C	1D	4870	JMP CHECK.DEFINITION	
			4880	-----	
			4890	PRINT.LINNUM.CHAIN	
9246-	A9	00	4900	LDA #0	reset counter to 0
9248-	85	16	4910	STA COUNTER	for each call
924A-	20	70	4920	.1 JSR TAB.NEXT.COLUMN	
924D-	A0	02	4930	LDY #2	point at line #
924F-	B1	9B	4940	LDA (TPTR),Y	
9251-	85	1D	4950	STA LINNUM+1	
9253-	C8		4960	INY	
9254-	B1	9B	4970	LDA (TPTR),Y	
9256-	85	1C	4980	STA LINNUM	
9258-	20	95	4990	JSR PRINT.LINE.NUMBER	
925B-	A0	01	5000	LDY #1	set up next pointer
925D-	B1	9B	5010	LDA (TPTR),Y	
925F-	F0	0B	5020	BEQ .2	end of chain
9261-	48		5030	PHA	
9262-	88		5040	DEY	
9263-	B1	9B	5050	LDA (TPTR),Y	
9265-	85	9B	5060	STA TPTR	
9267-	68		5070	PLA	
9268-	85	9C	5080	STA TPTR+1	
926A-	D0	DE	5090	BNE .1	...always
			5100		
926C-	60		5110	.2 RTS	
			5120	-----	
			5130	TAB.NEW.LINE	
926D-	20	8E	5140	JSR MON.CROUT	
			5150		

9270-	A9	07	5160	TAB.NEXT.COLUMN		
9272-	C5	24	5170	.1	LDA #7	first tab stop
9274-	B0	16	5180	.2	CMP MON.CH	cursor position
9276-	69	06	5190		BCS .3	perform tab
9278-	C9	21	5200		ADC #6	next tab stop
927A-	90	F6	5210		CMP #33	end of line?
927C-	E6	16	5220		BCC .2	
927E-	A5	16	5230		INC COUNTER	count the screen line
9280-	29	01	5240		LDA COUNTER	
9282-	F0	E9	5250		AND #1	look at odd-even bit
9284-	A9	8D	5260		BEQ TAB.NEW.LINE	both scrn and printer
9286-	20	F0	5270		LDA #CR	
9289-	4C	70	5280		JSR MON.COUT1	<CR> to screen only
			5290		JMP .1	...always
			5300			
928C-	F0	06	5310	.3	BEQ .4	already there
928E-	E5	24	5320		SBC MON.CH	calculate # of blanks
9290-	AA		5330		TAX	
9291-	20	4A	5340		JSR MON.PRBL2	
9294-	60		5350	.4	RTS	
			5360			
			5370			
9295-	A2	04	5380		PRINT.LINE.NUMBER	
9297-	86	1A	5390		LDX #4	print 5 digits
9299-	A9	30	5400	.1	STX LZFLAG	turn on leading zero flag
929B-	48		5410	.2	LDA #10	digit=0
929C-	38		5420		PHA	
929D-	A5	1C	5430		SEC	
929F-	FD	D2	5440		LDA LINNUM	
92A2-	48		5450		SBC PLNTEL,X	
92A3-	A5	1D	5460		PHA	
92A5-	FD	D7	5470		LDA LINNUM+1	
92A8-	90	0A	5480		SBC PLNTEH,X	
92AA-	85	1D	5490		BCC .3	less than divisor
92AC-	68		5500		STA LINNUM+1	
92AD-	85	1C	5510		PLA	
92AF-	68		5520		STA LINNUM	
92B0-	69	00	5530		PLA	
92B2-	D0	E7	5540		ADC #0	increment digit
			5550		BNE .2	...always
			5560			
92B4-	68		5570	.3	PLA	
92B5-	68		5580		PLA	
92B6-	C9	30	5590		CMP #10	
92B8-	F0	0C	5600		BEQ .5	zero, might be leading
92BA-	38		5610		SEC	turn off LZFLAG
92BB-	66	1A	5620		ROR LZFLAG	
92BD-	09	80	5630	.4	ORA #80	
92BF-	20	ED	5640		JSR MON.COUT	
92C2-	CA		5650		DEX	
92C3-	10	D4	5660		BPL .1	
92C5-	60		5670		RTS	
92C6-	24	1A	5680	.5	BIT LZFLAG	leading zero flag
92C8-	30	F3	5690		BMI .4	no
92CA-	E0	00	5700		CPX #0	if all zeroes, print one
92CC-	F0	EF	5710		BEQ .4	
92CE-	A9	20	5720		LDA #'	blank
92D0-	D0	EB	5730		BNE .4	...always
			5740			
92D2-	01		5750		PLNTEL .DA #1	
92D3-	0A		5760		.DA #10	
92D4-	64		5770		.DA #100	
92D5-	E8		5780		.DA #1000	
92D6-	10		5790		.DA #10000	
92D7-	00		5800		PLNTEH .DA /1	
92D8-	00		5810		.DA /10	
92D9-	00		5820		.DA /100	
92DA-	03		5830		.DA /1000	
92DB-	27		5840		.DA /10000	
			5850			
92DC-	AD	00	5860		CHECK.FOR.PAUSE	
92DF-	10	11	5870		LDA KEYBOARD	keypress?
92E1-	8D	10	5880		BPL .2	no, go on
92E4-	C9	8D	5890		STA STROBE	
92E6-	F0	0A	5900		CMP #CR	RETURN?
92E8-	AD	00	5910	.1	BEQ .2	yes
92EB-	10	FB	5920		LDA KEYBOARD	no, wait for
92ED-	8D	10	5930		BPL .1	another stroke
92F0-	C9	8D	5940		STA STROBE	
92F2-	60		5950	.2	CMP #CR	return .EQ. if RETURN
					RTS	

Speaking of Slow Chips.....Robert H. Bernard

William O'Ryan's article (AAL June 1984) about making the 65C02 work in II+s reminds me of some other slow chip problems I have had in the past with Apples.

Years ago, I had a problem with an SSM AIO card in an Apple that traced to a slow 74LS138 at position H2. The symptom was that every few hours the program would fly off into the weeds. I traced it to the device select for the slot, which caused the data on the bus to be late for ROM program fetches from the card. I was able to fix the problem in that case by swapping H2 with another '138 from a different (less critical) position.

Some time later I was able to fix a problem in another machine by swapping the ROM SELECT chip at position F12 (another 74LS138) with another '138. There are apparently many marginal timing situations in II+s, and they are not necessarily in the oldest ones.

All this slow circuit stuff has some interesting side effects. I personally had a number of conversations with SSM about this problem before I found the real cause, and all they could suggest was a capacitor on the clock line. Even after I found the problem, the SSM people I talked to seemed uninterested in the fix, perhaps because they couldn't apply it directly to their product.

The unfortunate end result was that a number of organizations that previously sold or recommended AIO cards stopped doing so. A domino effect was that our local retailer stopped pushing Anadex printers (which required the DTR signal, at that time only available on the AIO) rather than find another serial card to replace the AIO. I always wondered if the Anadex people noticed the effect on their sales....

Don Lancaster's AWiLe TOOLKIT

Solve all of your Applewriter™ IIe hassles with these eight diskette sides crammed full of most-needed goodies including . . .

- Patches for NULL, shortline, IIc detrashing, full expansion
- Invisible and automatic microjustify and proportional space
- Complete, thorough, and fully commented disassembly script
- Detailed source code capturing instructions for custom mods
- Clear and useful answers to hundreds of most-asked questions
- Camera ready print quality secrets (like this ad, ferinstance)
- New and mind-blowing WPL routines you simply won't believe
- Self-Prompting (!) glossaries for Diablo, Epson, many others
- Includes a free "must have" bonus book and helpline service

All this and bunches more for only \$39.95. Everything is unlocked and unprotected. Order from SYNERGETICS, 746 First Street, Box 809-AAL, Thatcher, AZ, 85552. (602) 428-4073. VISA or MC accepted.

Modify DOS 3.3 for Big BSAVES.....Bob Sander-Cederlof

Jim Sather (author of "Understanding the Apple II" and designer of the QuikLoader card) called today, and one topic of discussion was DOS 3.3's limit of 32767 for the maximum size of a binary file. Jim has been blowing 27256 EPROMs, which are 32768 bytes long. To write a whole EPROMs worth of code on disk it takes two files, because the EPROM holds one more byte than the maximum size file.

The limit doesn't apply if you write the file with the .TF directive in the S-C Macro Assembler, but it is checked when you type in a BSAVE command. The "L" parameter must be less than 32768.

I remembered that somewhere very recently I had read of a quick patch to DOS to remove the restriction. Where? Hardcore Computing? Call APPLE? Washington Apple Pi?

The answer was "yes" to both Call APPLE and W.A.P., because Bruce Field's excellent Apple Doctor column is printed in both magazines. The July 1984 Call APPLE, on page 55, has the answer:

"Sure, change memory location \$A964 in DOS from \$7F to \$FF. From Applesoft this can be done with POKE 43364,255. This changes the range attribute table in DOS to allow binary files as large as 65535 bytes."

By the way, please do not try to BSAVE 65535 bytes on one file. You will not succeed, because doing so will involve saving bytes from the \$C000-C0FF range. This is where all the I/O soft switches are, any you will drive your Apple and peripherals wild. And you will not be able to BLOAD it, because it will load on top of the DOS buffers. In general, do not BSAVE any area of RAM which includes \$C000-C0FF. Do not BLOAD into the DOS buffers or DOS variables.

If you want to test Bruce's patch, make the patch and then BSAVE filename,A\$800,L\$8E00. This will save from \$800 through \$95FF.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is \$18 per year in the USA, sent Bulk Mail; add \$3 for First Class postage in USA, Canada, and Mexico; add \$12 postage for other countries. Back issues are available for \$1.50 each (other countries add \$1 per back issue for postage).

All material herein is copyrighted by S-C SOFTWARE CORPORATION, all rights reserved. (Apple is a registered trademark of Apple Computer, Inc.)